Design and Implementation of a Micro Force Displacement System

A Senior Project

presented to

the Faculty of the Materials Engineering Department

California Polytechnic State University, San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Bachelor of Science

by

Evan Cate

June, 2013

Approval Page

Project Title: Design and Implementation of a Micro Force Displacement System

Author: Evan Cate

Date Submitted: June 7, 2013

CAL POLY STATE UNIVERSITY

Materials Engineering Department

Since this project is a result of a class assignment, it has been graded and accepted as fulfillment of the course requirements. Acceptance does not imply technical accuracy or reliability. Any use of the information in this report, including numerical data, is done at the risk of the user. These risks may include catastrophic failure of the device or infringement of patent or copyright laws. The students, faculty, and staff of Cal Poly State University, San Luis Obispo cannot be held liable for any misuse of the project.

Prof. Richard Savage _____

Faculty Advisor Signature

Prof. Richard Savage _____

Department Chair Signature

ABSTRACT

The design and implementation of a micro-force displacement system was completed to test various Micro-Electro-Mechanical Systems (MEMS) devices including silicon diaphragms and cantilevers. The system utilizes a World Precision Instruments Fort 10g force transducer attached to a World Precession Instruments TBM4M amplifier. A Keithley 2400 source meter provided data acquisition of the force component of the system. A micro prober tip was utilized as the testing probe attached to the force transducer with a tip radius of 5um. The displacement of samples was measured using a Newport M433 linear stage driven by a Newport ESP300 motion controller (force readings at constant displacement intervals). An additional 3 linear stages were used to provide X and Y-axis positioning of samples beneath the probe tip. The system components were mounted to an optical bench to provide stability during testing. C# was used to deliver the code to the individual components of the system. In addition the software provides a graphic user interface for future users that includes a calibration utility (both X/Y and force calibration), live force-displacement graph, motion control, and a live video feed for sample alignment. Calibration of the force transducer was accomplished using an Adam Equipment PGW153e precision balance to assign force values to the voltage data produced from the transducer. Displacement calibration involved the use of a microscope calibration micrometer. The transducer was characterized to provide a resolution of +/- 0.5 milligrams (4.9uN) with the ability to characterize samples with flexibility greater than 8.2372 mg/um. The displacement resolution of the system was determined to be 35 nm per step of the linear stages. The system was tested on a 4.4mm^2 diaphragm to characterize the force displacement of the device.

Keywords: AFM, Micro, Force, Displacement, System, MEMS, Diaphragm, Cantilever, Transducer

ACKNOWLEDGEMENTS

I would like to thank my partner Kevin Schapansky for his help with the software portion of this project. My advisor, Dr. Richard Savage for the opportunity to work on this project, his support, advice, and help obtaining funding. The Microsystems Technology Group, specifically Ross Gregoriev for his help with component selection, and Elizabeth Brooks for providing testing specimens for this project. Finally I would like to thank CP Connect for the funding that made this project possible.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

INTRODUCTION

PROBLEM STATEMENT

The Materials Engineering Department of Cal Poly does not currently posses the ability to analyze the mechanical properties of MEMS devices created within the Microfabrication Lab. Previous projects have used a surface profilometer for mechanical testing with limited success at the price of profilometer damage. The profilometer uses a fixed mass (the mass of the profilometer tip and arm) to apply force to the silicon cantilever MEMS, while a photodiode array measures the angular displacement of a mirror used in calculating the linear displacement of the profilometer tip. The profilometer tip and arm apply a 0.25mN force to the cantilever, with a profilometer resolution of 20nm. A micro force displacement system (MFDS) needs to be designed to perform a force displacement test on devices such as a 4.4mm by 4.4mm, 20um thick silicon diaphragm, or other such devices of this scale. The system must be robust enough to stand up to years of use, as well as be easily and economically repairable by students in the future. The system should improve upon the profilometer technique of testing, with a force equal or less than 0.25mN, with a resolution approaching 20nm.

MEMS FABRICATION

MicroElectroMechanical Systems (MEMS) are used in many industries for a wide array of applications. These devices are most often created from silicon, and utilize micro fabrication techniques to achieve device sizes on the micron scale. Some common uses for MEMS include pressure sensors, flow sensors, accelerometers, and micro-mirror arrays. Most of these MEMS devices rely on the physical actuation of silicon structures; for example accelerometers use the deflection of a cantilever beam to

measure changes in acceleration. Common uses of these devices include monitoring tire pressure using the deflection of a diaphragm to quantify pressure change, monitoring chemical reaction rates by tracking the change in flow of thermal mass in a flow sensor, monitoring the deflection of a silicon cantilever to quantify acceleration, or producing inkjet nozzles with via holes (Figure 1).

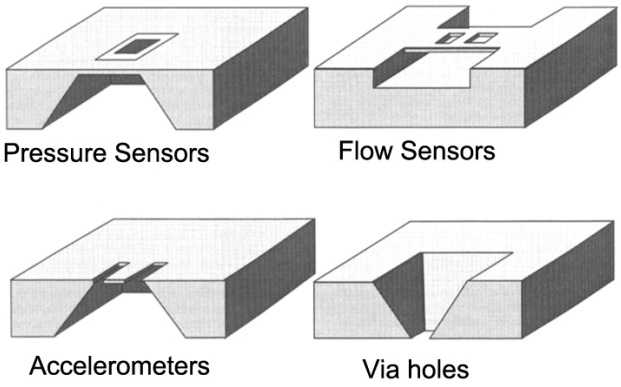


Figure 1: Examples of common MEMS devices.[1]

MEMS heavily utilize photolithography, surface micromachining, and bulk micromachining to form devices in a batch process, meaning hundreds or thousands of devices can be produced at once.[2] Photolithography is a process by which selective etching of a substrate can be accomplished (Figure 2). The first step is the application of a photoresist, which “after exposure to UV light and subsequent developing, remains on the substrate”.[3] Using a mask, the UV exposure can be contained to certain areas of the photoresist creating a pattern once the photoresist has been developed. After step (d) in Figure 2, steps (e) and (g) represent one process, while (f) and (h) represent a different process. Steps (e) and (g) are used for the deposition of materials onto the substrate through the use of physical vapor deposition (PVD), chemical vapor deposition (CVD), and spin coating (among other processes). Steps (f) and (h) are used to selectively etch

the substrate, in the case of silicon, an example of a common etchant is a HNO_3/HF solution.[3]

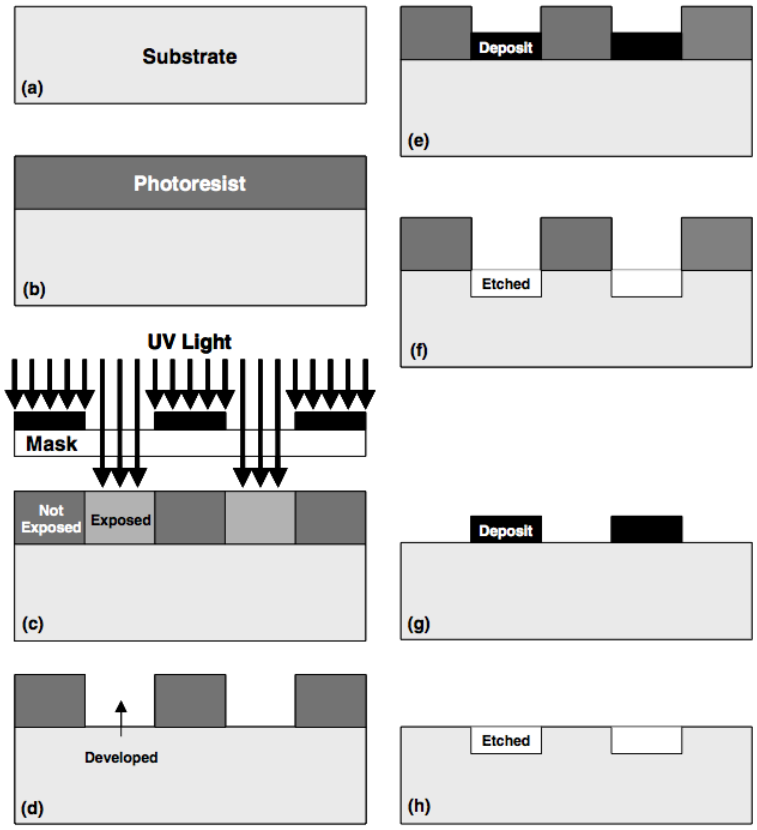


Figure 2: Example of a photolithography process used to create MEMS devices.[4]

As seen in step (h) of Figure 2, etchants can be used to selectively remove the substrate. Bulk micromachining can take this process further by utilizing the crystallographic planes of single crystalline silicon. Through the use of anisotropic etchants such as potassium hydroxide (KOH) or tetramethylammonium hydroxide (TMAH), a sidewall angle of 54.74˚ can be achieved, allowing for structures such as through holes, membranes (for diaphragms), and v-grooves to be created (Figure 3).[5]

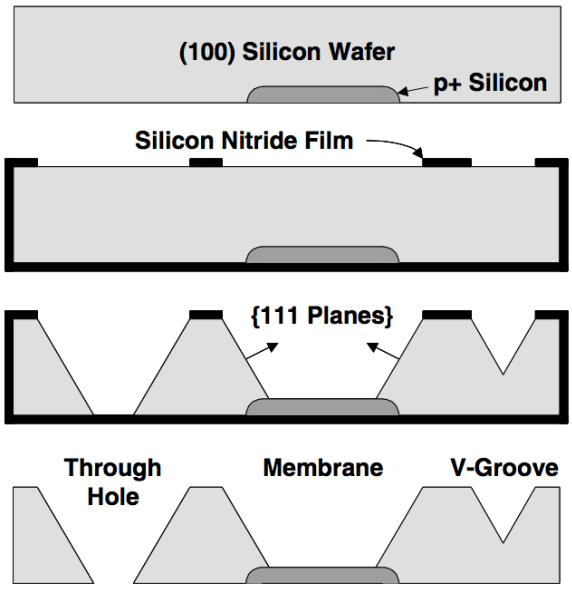


Figure 3: Example of bulk micromachining to create through holes, membranes, and v-grooves.[4]

Another important MEMS processing technique is surface micromachining, in which films for structural elements are deposited onto the surface of the substrate. Advantages of surface micromachining include higher yields (less waste due to set sidewall angle), and both lateral and vertical movement of components.[5] Figure 4 demonstrates the creation of a cantilever for potential use as an accelerometer. First a sacrificial layer is deposited onto the substrate, next a structural layer (deposited using low pressure chemical vapor deposition (LPCVD) in the case of polysilicon) is deposited, and finally the sacrificial layer is removed from beneath the structural layer, leaving a freestanding structure.[5]

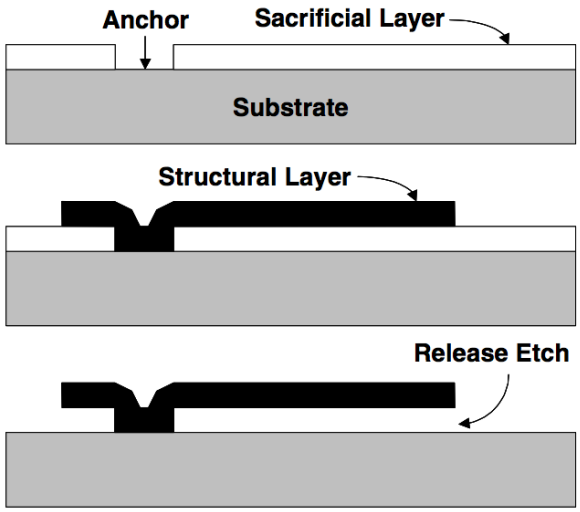


Figure 4: Example of surface micromachining to create a MEMS cantilever.[4]

Through the combination of these various techniques, a wide array of MEMS devices can be created. In a production setting, the device yield is high due to excellent process control and elevated cleanroom standards. Within the Microfabrication Lab at Cal Poly, lower cleanroom standards and less process control (due to fewer batches being processed) leads to lower device yield, and the need to test and inspect individual MEMS devices.

MOTIVATION

Testing of final packaged devices (packaging refers to the material added to the MEMS device to protect it from environmental as well as electrical contact) is important to calibrate the devices prior to shipping to customers. Take an accelerometer for example, mechanical testing in the form of known accelerations are applied to the final device. The output of the accelerometer is then calibrated to the known acceleration to ensure proper function.[6] These tests are easily performed on packaged devices.

Packaging and testing of all the final devices produced in the Microfabrication Lab would be prohibitively expensive and tedious process due to the additional equipment and labor required. A low device yield equates to throwing away many of the final packages produced due to bad MEMS devices within the packaging. Quantifying the physical properties of MEMS devices prior to packaging allows selective packaging of only properly functioning MEMS devices produced in the Microfabrication Lab. In addition, testing prior to final packaging allows the user to validate processing steps in addition to comparing the physical testing results to theoretical models.

Validating processing steps helps to ensure that proper etch rates are being observed for specific steps of processing. New etchants, old etchants, or etchants at different temperatures all produce different etch rates. If an etch rate is too slow, devices could be thicker than originally thought, and vice versa for faster etch rates. Quantifying this change through the observation of mechanical properties (deflection of a diaphragm etc.) allows the user to correct the processing step before moving to the subsequent step in processing.

Often theoretical models are not accurate representations of MEMS devices due to thin film stresses introduced during processing (such as stresses introduced from high temperature deposition of materials and consequent cooling to room temperature).[7] Physical testing of MEMS devices in the lab allows for a more accurate characterization of actual devices created. Identifying the differences between theoretical models and physical results can help to narrow the differences between the two, creating more accurate models in the future as well as a more complete understanding of material properties such as Young's Modulus.

Other systems can be used to quantify the relationship of force and displacement of MEMS devices (Table 1). On campus an Ambios Profilometer was used to apply a fixed force (the mass of the profilometer tip and arm) to MEMS devices to observe the displacement produced (Figure 5).[2] The downside of this system is that if the force applied by the profilometer exceeds the capacity of the device, the device will break without data being collected. In addition the fixed force only gives one data point, not multiple force and displacement measurements, making the creation of a force displacement graph impossible. The profilometer was not designed to carry out force displacement measurements; as such the risk of damaging the profilometer is higher than that of a dedicated Microtester system.

Table 1: Past Force Displacement Systems.[2,8]

System	Force Resolution	Displacement Resolution
Ambios Profilometer	0.28 mN (fixed)	20nm
Instron 5948 Microtester	<20 mN	20nm

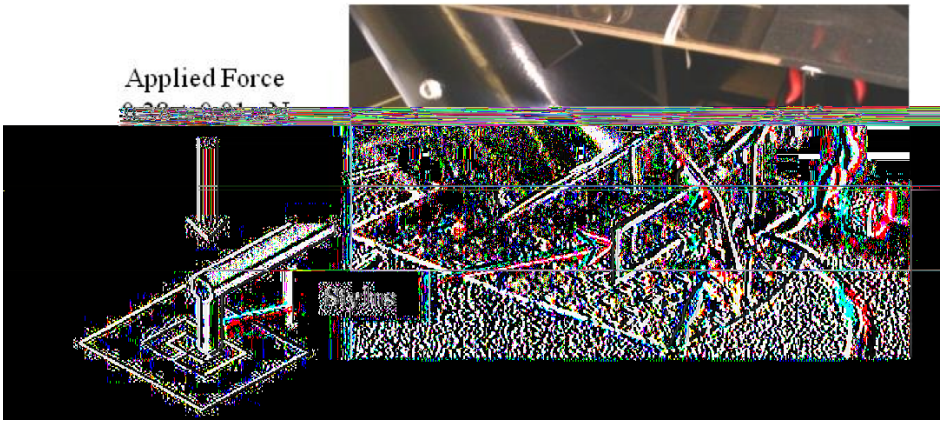


Figure 5: Ambios profilometer used for force displacement tests.[2]

The commercial side of force displacement testing at this scale is the Instron 5948 Microtester (Figure 6), which has a poor force resolution (Table 1), but comparable displacement resolution to the Ambios profilometer. One main issue with the Instron system is the mounting hardware involved to test a device. Wafer level devices would require a special mounting bracket to be tested.

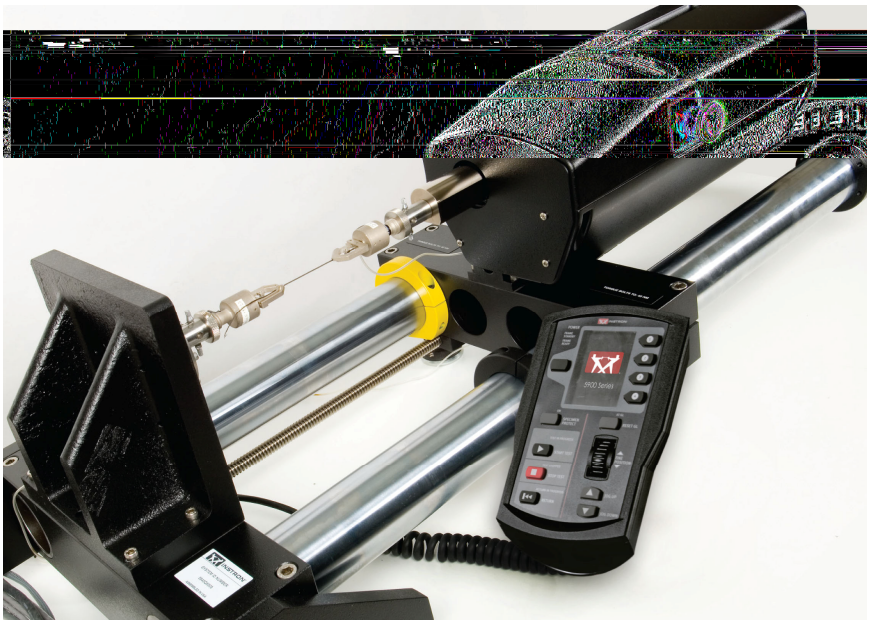


Figure 6: Instron 5948 Microtester system.[8]

Both of these systems are capable of recording force displacement data for MEMS devices created in the Microfabrication Lab, however a new system specifically designed for the devices created on campus would make it easier for students to collect data from MEMS devices.

Manufacturability:

One disadvantage of commercial force-displacement measurement systems such as the Instron 5948 is maintenance and consumables costs. Testing specimen fixtures as well as additional jigs represent supplementary income for companies like Instron, which for an educational institution signifies extra budget allocations for testing. Manufacturability of the MFDS was a chief concern, especially in the current budget crisis. The MFDS was designed to use mostly off the shelf components and standard fasteners, with the exception of two parts (both easily manufactured on campus). Additional testing specimen fixtures for the MFDS can be created using the rapid prototyping machine in the Materials Engineering Department, further reducing the cost of system.

Economics:

Testing MEMS devices during production allows the Materials Engineering Department and students to observe mechanical properties at key steps in production. If a certain process is not producing the desired effects, the MEMS devices can be reprocessed, reducing the probability of non-functioning devices being produced. Reducing the total number of devices produced for a given project saves the Materials Engineering Department money on process consumables including sputtering targets, photolithography supplies, and etchants among other supplies. Due to the hazardous nature of transporting some of these supplies, the cost of microfabrication can be extremely expensive, thus the economic benefit of producing the MFDS outweighs the capital cost of the system.

DESIGN

HARDWARE

The MFDS incorporated both new equipment, and equipment used in other past projects in the Materials Engineering Microfabrication Lab. A systems block diagram was used to map system components (Figure 7).

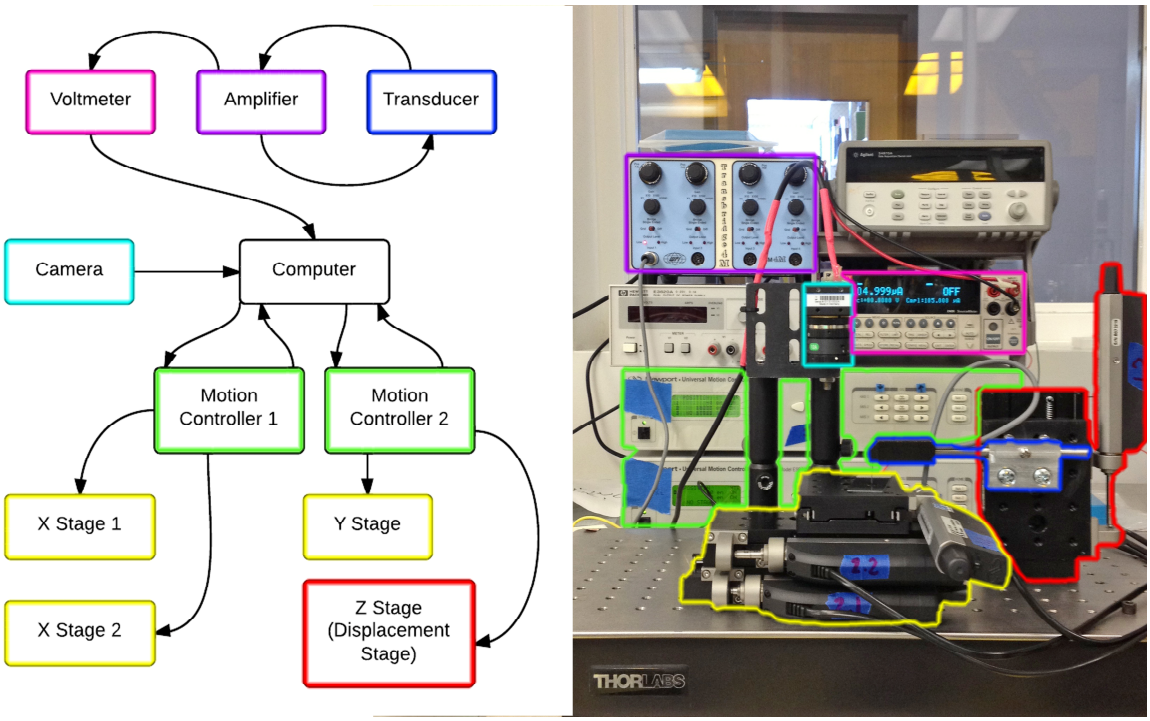


Figure 7: System block diagram of hardware design. Note: color-coding of blocks to hardware components.

A breakdown of components and associated costs was compiled (Table 2), using the costs to purchase a new component in the case of used equipment. The total approximant cost of the components for the system is $21,078.

Table 2: Components and costs of MFDS

System Component	Actual Component	New/Used	Total Cost
Computer	Dell	New	~$500
Camera	The Imaging Source DMK 72AUC02 camera with M0814-MP2 lens	New	$668
Motion Controllers (2)	ESP300	Used	~$5,800
X Stages (2)	Newport M443 & LTA-HS	Used	~$3,800
Y Stage	Newport M443 & LTA-HS	Used	~$1,900
Z Stage	Newport M443 & LTA-HS	Used	~$1,900
Voltmeter	Keithley 2400	Used	~$4,300
Amplifier	World Precision Instruments TBM4M	New	$1,895
Transducer	World Precision Instruments FORT10G	New	$315
		Total	~$21,078

The transducer and amplifier were the two most important new items to the system, allowing for the acquisition and processing of force observed during testing. Multiple options were considered to measure the force of the system, among those considered was using a constant weight while looking at displacement with capacitance, an AFM style transducer probe, a piezoelectric bending actuator, and a piezoelectric force transducer. Of these options, the piezoelectric force transducer was chosen for its cheap relative cost, accuracy, and resolution.

The transducer subassembly of the MFDS consisted of three parts; the transducer body, a vented #2-56 screw, and a micro-prober tip (Figure 8). The utilization of a vented screw to connect the transducer to the micro-prober tip allowed future users to interchange tips based on the application during testing. The micro-prober tip was epoxied into the vented screw, which was then attached to the transducer body.

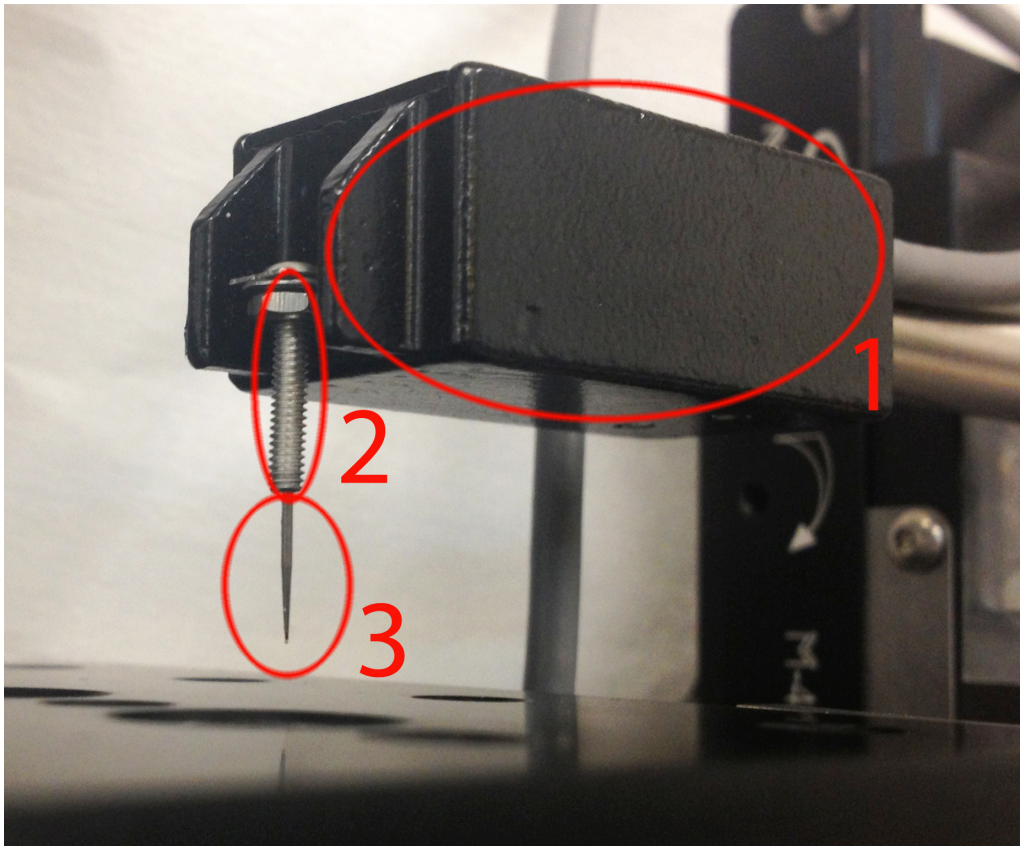


Figure 8: Transducer Subassembly: 1) Transducer body 2) Vented #2-56 screw 3) Micro-prober tip. Interchangeable micro-prober tips allow future users the ability to tailor the system to their needs.

To load the micro-prober tip attached to the vented screw, a rapid prototyped ABS fixture was created to hold the transducer in place to ensure that the load limit of the transducer was not exceeded during attachment (Figure 9). The vertical post seen in Figure 9 mated with the bottom of the transducer cantilever, lending extra support to remain below the 20g limit of the transducer.

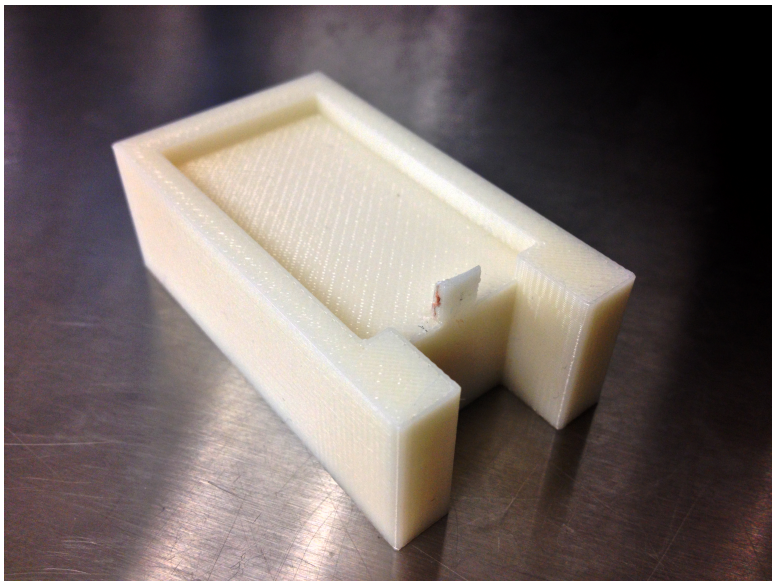


Figure 9: Rapid prototyped transducer loading fixture. This fixture allowed for transducer stability during tip loading which ensured the transducer did not exceed the specified load limit.

SOFTWARE

The software for this project was created with the help of Kevin Schapansky (CSC) who converted the functional requirements of the software into an integrated system. Initially LabView was chosen for the development environment for the project. However, due to compatibility issues with certain components (namely the ESP300 Motion Controllers), LabView could not be utilized. C# provided the best alternative, with the capability to create a user-friendly graphic user interface (GUI). LabView provided built in functions available to the user, however in C# individual RS-232 commands were sent to each of the hardware components. One component exception to communicating through RS-232 was the camera, which had a C# library of functions available which allowed higher-level component integration into the GUI.

The RS-232 commands utilized in the MFDS were unique to each component, with software written for the motion controllers, voltmeter, and analytical balance. Examples of RS-232 commands and their functions are shown in Table 3. For a complete list of commands used, reference Appendix 1.

Table 3: RS-232 Commands for Individual Hardware Components

Component	Machine Code	Function
ESP 300 Motion Controller	1PR10	Move axis 1 10 positive units relative to the current position
Keithley 2400	:READ?	Read voltage
Analytical Balance	!KP	Read balance

After the integration of the components into the software, a GUI was created to serve as an interface for the end user (Figure 10). The interface allows full control of the system components as well as testing parameters.

Figure 10: Software GUI overview. See Appendix II for a larger view of the GUI

The software can be broken down into individual components and the respective functions of each. The first section of the software is the image from the camera and camera controls (Figure 11). The camera view allows the user to see the sample being tested as well as zoom in or out on the sample using the slider beneath the image. The red crosshairs visible in the center of the camera view represent the portion of the sample on which the test will take place. The slider provides a reading of the zoom level of the camera; Figure 11 shows the current zoom level at 100%.

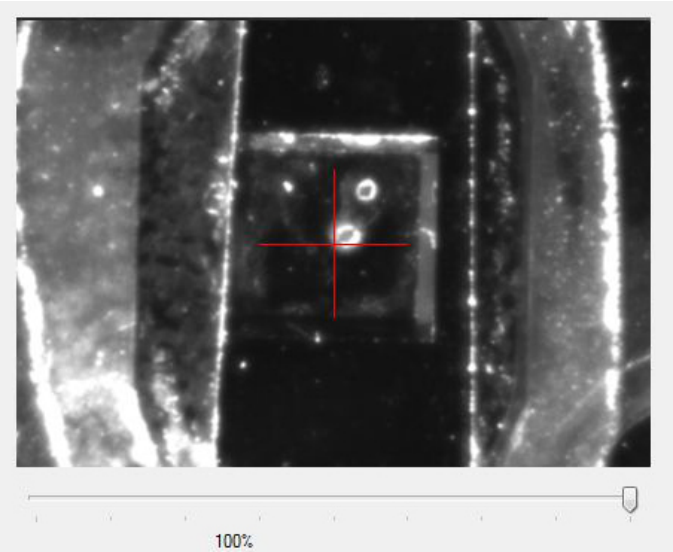


Figure 11: Software camera integration and image processing.

Beneath the imaging control is the motion control section of the GUI (Figure 12). This section allows the user to control the 4 total stages of the system. The two X-axis stages are controlled using the “Left” and “Right” buttons as well as the top slider labeled “Step Size”. The Y-axis stage is controlled using the “Up” and “Down” buttons as well as the “Step Size” slider. The Z-axis stage is controlled using the “Probe Up” and “Probe Down” buttons as well as the “Vertical Step Size” slider. It should be noted that the “Step Size” slider controls the motion step size for both the X and Y-axis stages. Encoder units (en) were chosen as the displacement step sizes, which each represent approximately 35nm of displacement per step.

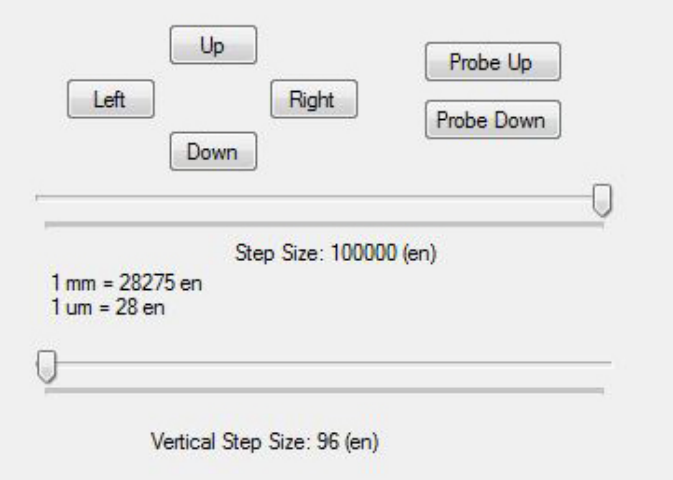


Figure 12: Software motion control of X Y and Z axes.

The testing is initiated through the software and allows the user to input various test parameters designed to protect the test samples as well as the transducer of the MFDS. The testing parameters include the stopping voltage on the transducer, reading delay, and step size (Figure 13). In addition the testing parameter section allows the user to define the output filename and location. The live force and voltage the transducer is experiencing is displayed as a preventative measure against breaking the transducer. Should the force on the transducer approach the force limit, the test can be stopped utilizing the “Stop Test” button. The transducer force limit is 10 grams, however the user should not exceed 5 grams of force during testing (~8V).

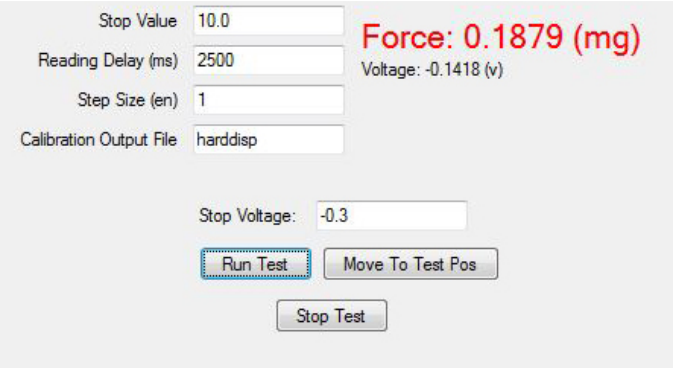


Figure 13: Software testing parameters as well as live transducer force and voltage readings.

Once the test has been initiated, a live graph displaying the force and displacement of the test is produced (Figure 14). The graph is not for data processing purposes (it does not take into account systemic displacement) but is instead used as a testing metric to ensure the transducer does not experience excessive force. In addition the graph is also used as a tool to stop tests once the sample has broken. In Figure 14 the sharp drop in force at approximately 350 en represents the loaded sample being broken. Once this behavior is observed in the graph, the user should stop the test to prevent damage to the transducer.

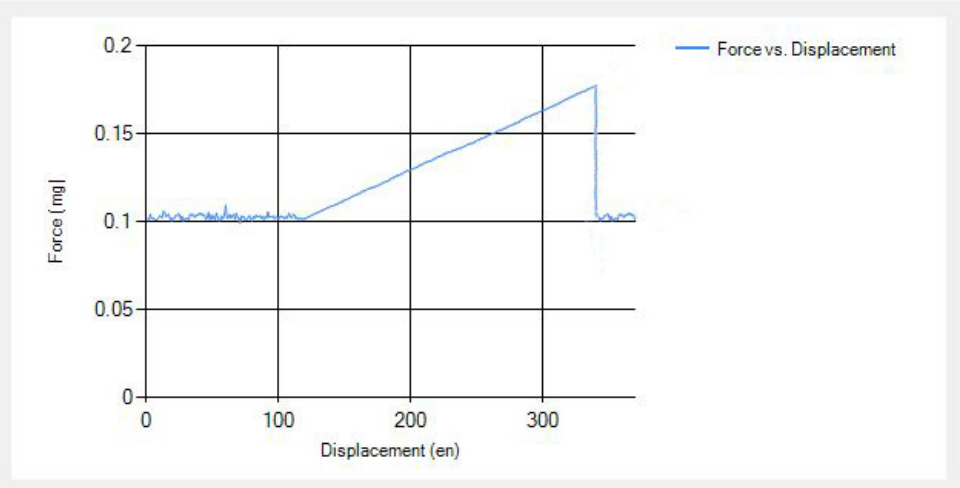


Figure 14: Software live force displacement graph

Prior to any testing, the system must be calibrated in several ways. For a complete explanation of system calibration, see the CALIBRATION section. The calibration tools included in the software encompass calibration for the transducer, and the X and Y-axis offset calibration (Figure 15). The “Balance/Keithley Calibration” allows the user to characterize the voltage and force produced when the transducer is brought into contact with an analytical balance. The “XY Calibrate” button allows the user to define the offset from the crosshairs on the camera to the actual tip of the transducer. The assumed X and Y-Axes offset represents the current offset (in encoder units) that the system perceives is correct. Once the user has realigned the X and Y-Axis (see X &

Y AXIS CALIBRATION section) the actual offset reflects any changes from the initial offset the system has generated. As visible in Figure 15 the actual and assumed offset for both the X and Y-Axes are presented to the user for reference. Using the “Backtrack” button, the point source used for calibration can be moved from the transducer to the camera (and vice versa) to ensure the X & Y-Axis calibration was successful.

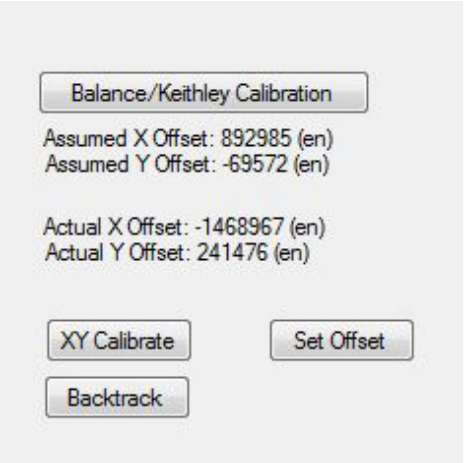


Figure 15: Software calibration tools

Once a test is completed, the system outputs a comma separated value (CSV) file with a header indicating the date, time, and testing parameters of the test (Figure 16). The CSV file only outputs the displacement in encoder units (en) and the voltage observed, both of which must be converted to physical units using equations found through calibrating the z axis displacement, systemic displacement, and transducer force.

```
Date: 5/26/2013 2:23:04 PM Stop Voltage: -3 (v) Reading Delay: 1000 (ms) Step Size: 1 (en)
1,-2.065006E+00
2,-2.066332E+00
3,-2.069512E+00
4,-2.075897E+00
5,-2.084172E+00
6,-2.084043E+00
7,-2.084508E+00
8,-2.086334E+00
9,-2.086414E+00
10,-2.086228E+00
```

Figure 16: Example of output CSV file.

CALIBRATION

X & Y AXIS CALIBRATION

In order for the user to utilize the cross hairs of the camera view to align samples, the offset from the center of the camera to the tip of the transducer must be known by the system. To calibrate the X and Y stages, a small amount of silicone grease was placed on the transducer tip. A glass microscope slide was then placed onto the testing specimen area. The tip of the transducer was brought into contact with the glass slide, and then lifted off. The “XY Calibrate” button was then selected. At this point the software used the assumed offset to bring the silicone grease “dot” close to the center of the camera view. The X and Y stages were then used to position the dot under the crosshairs of the camera. Once satisfied with the position, the “Set Offset” button was selected. Through the course of calibration, the software tracked the movements made by the user and added or subtracted the displacement values from the assumed offset to produce the actual offset. Once the calibration was complete, a sample could be placed into the MFDS and aligned using the crosshairs. The “Move To Test Pos” button could then be pressed to move the sample from the camera crosshairs to the transducer tip. Repeating this process multiple times insured that the x and y stages returned to the same position each time during calibration.

Z AXIS DISPLACEMENT CALIBRATION

The resolution of the linear stages of the MFDS was specified as 35.367 nm. To ensure that this displacement specification was correct, the Z stage was calibrated using a reticle calibration stage micrometer, with 100um increments. The Z stage was placed below the camera, and then the stage was moved 28281 en units (equal to 1mm or 10

steps of the reticle calibration stage micrometer). The stage aligned exactly with the 1mm mark of the reticle micrometer, ensuring that the encoder units specified were correct. The displacement data was then used to convert the raw output encoder units to displacement in microns.

SYSTEMIC DISPLACEMENT CHARACTERIZATION

The transducer used in the MFDS flexed as it created a voltage. The displacement produced within the transducer (and anywhere else in the system) was characterized to subtract the systemic displacement from the total displacement. The systemic displacement was characterized in the form of a relationship between a voltage value and the corresponding displacement value. The equation representing this relationship was obtained by lowering the transducer onto a rigid surface. The rigid surface chosen was a glass microscope slide mounted in the testing specimen area. As the transducer was lowered onto the microscope slide, an increasing amount of voltage was produced as the displacement increased (as expected). 5 samples were collected with the same dwell time (2000ms) and step size (1en). Tests were also conducted at a dwell time of 1000ms to ensure that the transducer voltage produced did not change based on a change in dwell time. The end result of the process was a graph displaying the displacement observed for a specific voltage value (Figure 17).

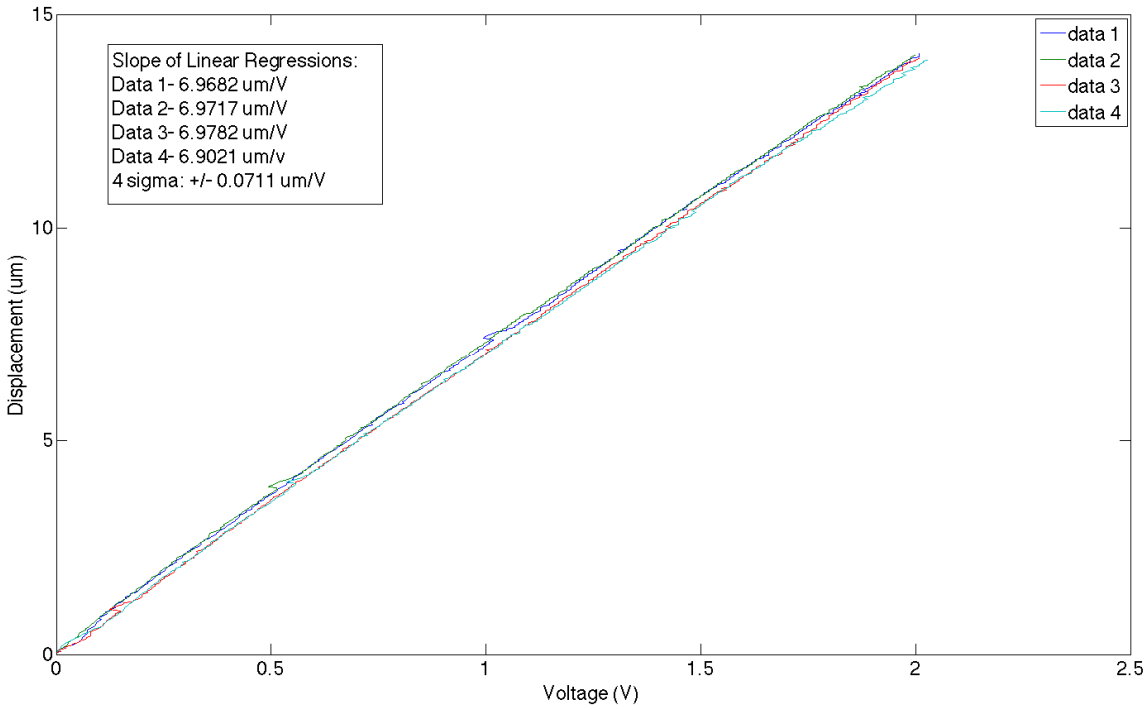


Figure 17: Graph of voltage displacement data for systemic displacement characterization.

The relationship of voltage and displacement from Figure 17 was linear in nature, with the slope of the line the most important element (the y intercept did not matter due to the data being normalized to zero as part of the data processing). The average slope of the systemic displacement calibration tests was 6.9551 um/V, or 0.14378 V/um, meaning that for each volt recorded, approximately 6.9551 microns of displacement can be attributable to systemic displacement, not sample displacement. +/-0.0711 um/V encompassed a 95% confidence interval (4 sigma) of the average slope of the systemic displacement calibration. For accurate data to be produced, the displacement of the sample should exceed 6.9551 um/V.

TRANSDUCER FORCE CHARACTERIZATION

The raw voltage output from the Keithley 2400 voltmeter must be translated into force to process the data from the MFDS. To accomplish the force calibration, the

transducer is lowered onto an analytical balance. As the transducer displacement increases, voltage data from the transducer, as well as force data from the analytical balance is collected. The relationship obtained from the force and voltage data allows the raw output voltage from the transducer to be translated into a force measurement.

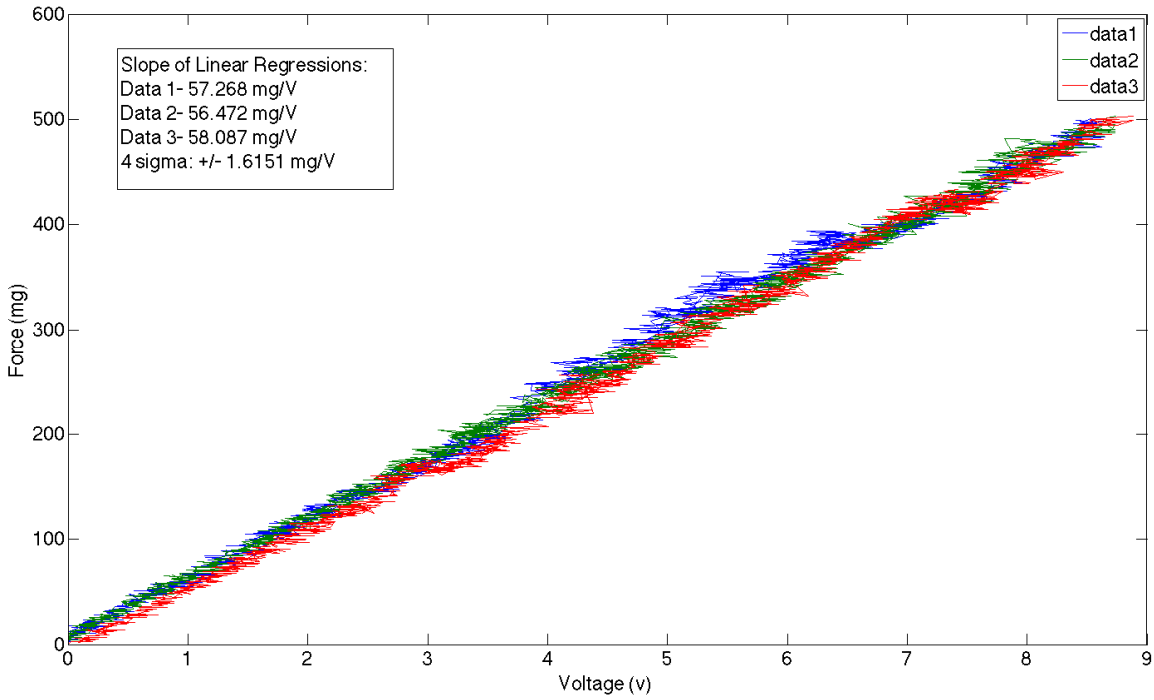


Figure 18: Graph of transducer force calibration, voltage generated by the transducer (x axis) versus force observed on the analytical balance (y axis).

The linear regressions from 3 tests were obtained, and then a 95% confidence interval was created using the slopes of the lines. This allows for 95% confidence as to the force experienced by the transducer for a certain voltage output. The data suggests that tests will be accurate to within +/- 1.6151 mg/V force with 95% confidence with 1000X amplification of the transducer output signal. With the resolution of the Keithley 2400 voltmeter at 1uV, each data point should be accurate to within +/- 1.6151 ng of force at full resolution. The systemic noise of the system was in the milligram range, suggesting that the force calibration should not be a limiting factor in testing.

Applying the minimum displacement result from the systemic displacement calibration, a new value of 8.2372 mg/um of displacement represents the stiffness of the transducer, samples tested should be less stiff than the transducer stiffness.

CYCLIC LOADING CHARACTERIZATION

An issue observed during calibration was drift between samples. Over time however, the drift subsided to negligible levels. To characterize the drift, the transducer was brought into contact with a glass microscope slide until the transducer registered approximately 2 volts (~100mg). Next, data was collected once per second for approximately 25 minutes, finally the transducer was lifted off the glass microscope slide, left for 25 minutes, then the test was repeated. After 3 iterations, the data was collected and plotted (Figure 19).

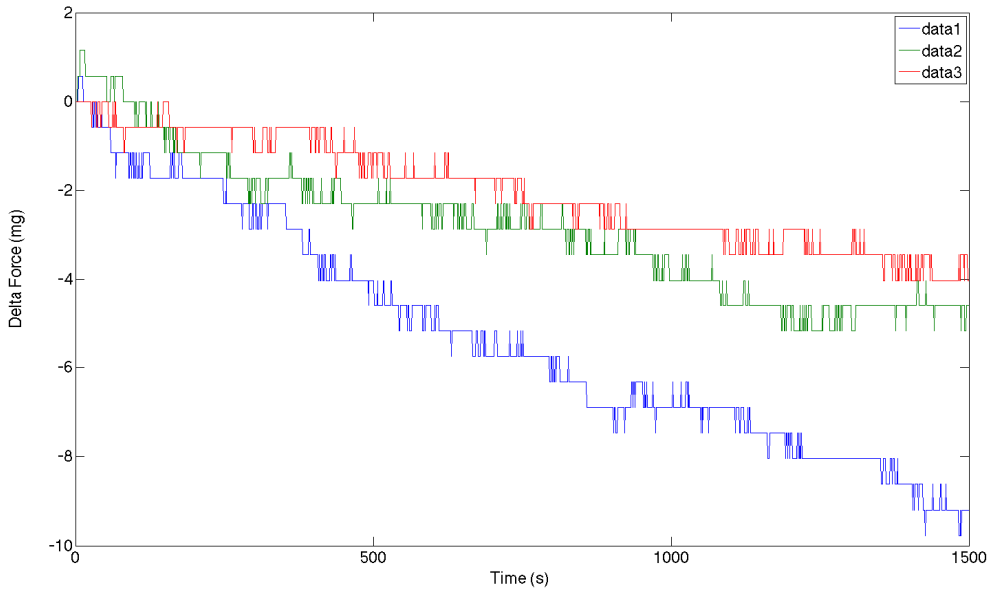


Figure 19: Cyclic loading of transducer to characterize the warm up time of the system.

The first sample (data 1) drops 2 milligrams of force in the first 60 seconds, suggesting that data collected when the system is first turned on could be skewed beyond the confidence of the force characterization previously performed. By the third

sample (data 3) the change in force does not exceed +/- 0.5 milligrams until after 7 minutes, and stays at zero for 10 seconds at the beginning of the test, allowing for accurate transducer measurements. As seen in Figure 19, the data suggests that the limiting resolution of the system is closer to +/- 0.5mg (+/-4.9uN) produced from system noise. From this data, the system should be allowed to warm up for at least 1.5 to 2 hours prior to testing MEMS devices.

Testing was performed on a 4.2 X 4.2 mm silicon diaphragm with a thickness of 20um. The system was allowed to warm up for 2 hours before testing was executed. Three trials were performed with 2000ms dwell time between each reading, and a 1 encoder unit displacement step size. The test was programed to stop at 20 mg of force, with the three tests averaged together to eliminate noise.

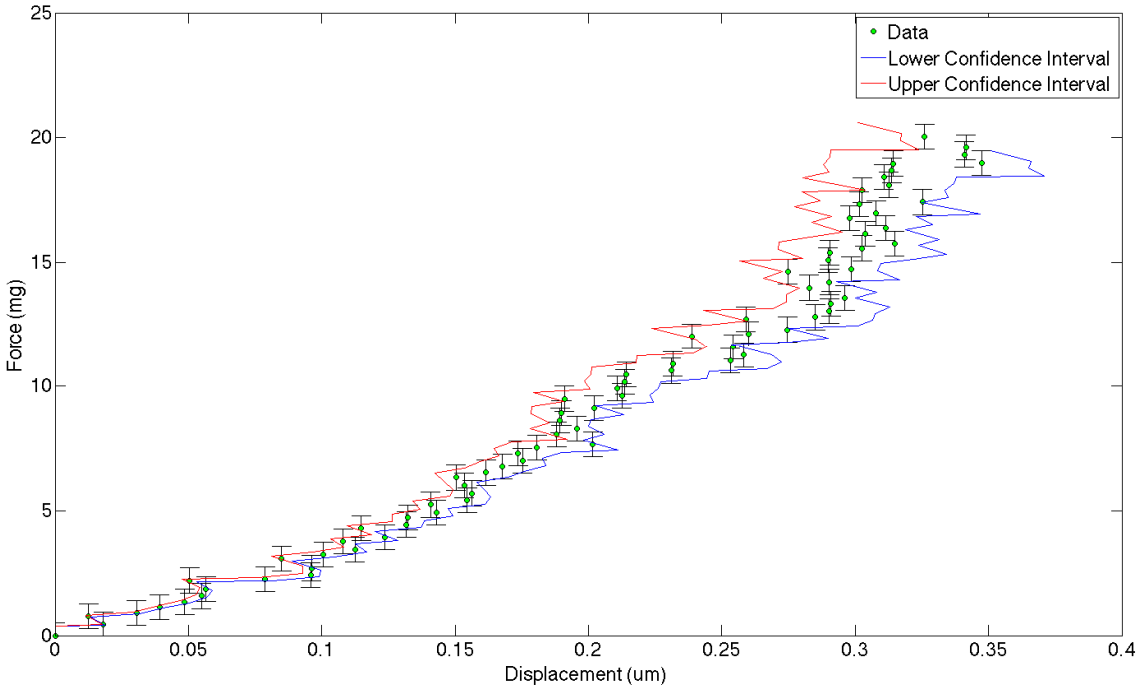


Figure 20: Force displacement graph for 4.2mm x 4.2mm silicon diaphragm. The green dots represent the actual data collected, with the error bars encompassing +/- 0.5 mg of each data point. The red curve represents the upper bound of the test confidence, and the blue curve represents the lower bound of the test confidence.

The graph of the results incorporates the upper and lower bound of the confidence produced through both the systemic displacement characterization, and the transducer force characterization. The red line was created using the upper force and displacement confidence values for slope, while the blue line utilized the lower force and

displacement values for the slope. The error bars on the data line represent the resolution of +/-0.5 mg as determined through the cyclic loading characterization.

DISCUSSION

CALIBRATION

The characterization of the force transducer for this project presented many problems. The sources of noise within the system has not yet been fully quantified, potential sources of error include vibrations, analytical balance errors, thermal effects, and assumptions during systemic displacement characterization.

Vibrations contributed to the cancellation of multiple testing attempts, evidence of vibrations can be seen in several test runs of Figure 17, with bumps in the data produced from people walking by the Microfabrication Lab, or slamming doors. The location of the MFDS on the top of a stainless steel table does not isolate the system from vibrations; in some cases the location amplifies the vibrations present.

The analytical balance used for testing could have produced inaccurate force data if the transducer tip had not been lowered normal to the surface of the balance, or if the tip was brought in contact with the surface too rapidly, not allowing the balance to settle and display accurate force values.

The affect of temperature on calibration was not addressed in this project. Variations in temperature in the Microfabrication Lab could affect the calibration curves of the MFDS, invalidating tests performed at temperatures inconsistent with the calibration temperature.

One source of error assumed to be negligible was the assumption of complete rigidity of the glass microscope slide used for systemic displacement calibration. For calibration, it was assumed that the only non-rigid body of the system was the piezoelectric cantilever used for force measurements in the transducer body (Figure 21).

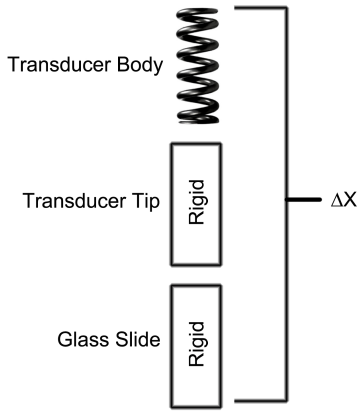


Figure 21: Assumption of rigidity for systemic displacement calibration.

A closer approximation of the displacement present in the system would account for the displacement of the transducer tip as well as the displacement of the glass slide (Figure 22). Due to the relative difference in the stiffness of the glass slide and transducer cantilever, the glass slide was assumed to be rigid. In addition, shear forces between the vented screw and transducer tip was assumed to be negligible because the area over which the shear forces were present was much larger than the forces experienced during calibration and testing. Quantifying the stiffness of both the glass slide and transducer tip would eliminate potential displacement error.

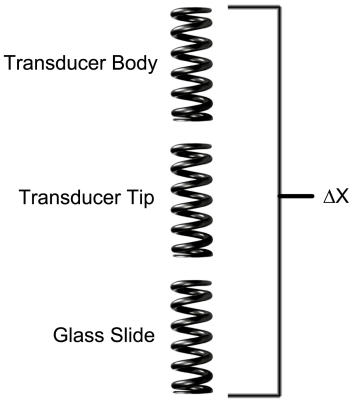


Figure 22: More accurate representation of sources of displacement within the MFDS system.

The validation of the test results was accomplished by comparing the test results to theoretical results derived through the use of lower and upper bounds of stiffness values for the diaphragm tested. Modeling the diaphragm as a fixed-fixed flexural beam resulted in the lower bound of the test (minimum stiffness of the diaphragm). By leaving two of the diaphragm edges unconstrained, the fixed-fixed flexural beam results will model the force displacement curve representing the lower bound of diaphragm stiffness. First the moment of inertia was calculated, using the diaphragm width of 4.4mm (w) and the thickness of 20um (t) (Equation 1).[9]

$$I = \frac{wt^3}{12}$$

Equation 1: Determination of moment of inertia for fixed-fixed flexural beam.[9]

Next Equation 2 was used to represent the displacement produced from a specific force value applied to the fixed-fixed flexural beam. The E value for intrinsic silicon was determined to be 1.5e11 Pa.[9] The L value was equal to the side length of the diaphragm (4.4mm).

$$Z = \frac{FL^3}{48EI}$$

Equation 2: Determination of deflection of fixed-fixed flexural beam based on an applied force.[9]

Equation 2 was then modified to provide force values for a given displacement value (the same displacement values produced during the testing with the MFDS), in addition the equation was converted from newtons to milligrams of force (Equation 3). Equation 3 represents the lower bound of stiffness for the diaphragm tested.

$$F = \frac{ZEI}{2124.49L^3}$$

Equation 3: Determination of force applied on fixed-fixed flexural beam for a certain deflection value, F in milligrams.

The upper bound of stiffness was modeled by distributing the force applied by the transducer over the entire area of the diaphragm (4.4mm^2). The equation for the displacement of the center of a diaphragm (Equation 4) was modified to analyze the force needed to produce a specific displacement value (Equation 5).

$$Z = \frac{0.0138a^4P}{Et^3}$$

Equation 4: Maximum deflection of the center of a fixed square plate.[9]

Equation 4 was modified to convert the force units from newtons to milligrams, as well as convert the pressure from Equation 4 to a force (Equation 5).

$$F = \frac{ZEt^3}{1407.21a^2}$$

Equation 5: Determination of force applied on the center of a fixed square plate for a certain deflection value, F in milligrams.

The lower and upper limits (Equation 3 and 5) respectively were then plotted over the same displacement range as the test results of the diaphragm tested (Figure 23). The observed data from the MFDS testing falls within the lower and upper limits of stiffness for the diaphragm, signifying that no source of error in the system affected the data beyond the theoretical limits specified.

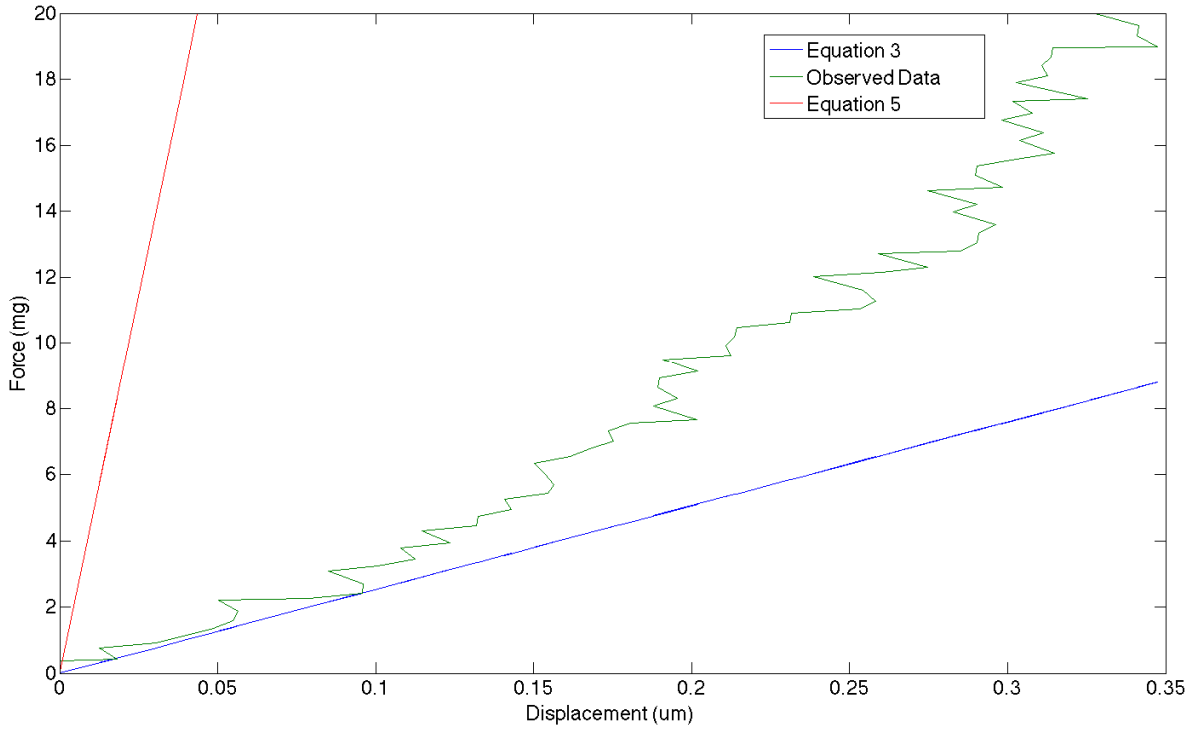


Figure 23: Lower and upper theoretical limits of stiffness for diaphragm tested.

The diaphragm was expected to display stiffness higher than that of the fixed-fixed flexural beam because the two sides left unconstrained in the fixed-fixed flexural beam model would be fixed, creating additional support for the diaphragm, increasing stiffness. In addition, the diaphragm was expected to exhibit lower stiffness than the results of Equation 5, due to the low resultant pressures generated for the force values. Figure 23 confirms that the MFDS collected data from the tested diaphragm, not noise.

RECOMMENDATIONS FOR FUTURE WORK

The MFDS system could be improved in the future through the addition of a vibration isolation table, the use of a more accurate analytical balance, further statistical analysis, additional device mounting hardware, a camera ring light, and software improvement.

A vibration isolation table would eliminate noise from other equipment present in the Microfabrication Lab. The location of the MFDS on a stainless steel table transfers noise from surrounding equipment as well as people moving near the MFDS. A vibration isolation table would allow the use of the MFDS during high traffic times of the day without data errors from vibrational disturbances.

Utilizing a more accurate analytical balance would allow more accurate displacement equations to be produced for use during testing. This would lead to an increase in resolution allowing for the testing of smaller structures.

A statistical analysis of the repeatability and reproducibility (Gauge R&R study), as well as a closer look at the statistical resolution of the system would ensure that accurate data is collected from the system. A sample of known modulus (such as an AFM tip) should be used to additionally quantify the accuracy of the MFDS system.

The addition of a camera ring light would make selecting the test site on the testing specimen faster and more accurate. If additional extension tubes were used on the camera to increase the magnification of the lens, the ring light would be necessary to provide enough light to image the testing specimens.

Some software improvement could be accomplished to provide additional functionality to the user. A micrometer on the camera view of the software would allow the user to quantify the size of the testing specimens. Updating the live force displacement graph by accounting for the systemic displacement would provide a more accurate picture of the testing conditions to the user.

CONCLUSIONS

The MFDS device achieved displacement resolution of approximately 35 nm, in the same order of magnitude as the other two systems being used for measurements of this scale. Without the replacement of the linear stages used in the system, the displacement resolution cannot be improved.

The force transducer of the system presented several challenges to the overall functionality of the system. The current resolution of the system is +/- 1.6151 nanograms based on full resolution of the Keithley 2400 voltmeter however noise in the system suggests that the resolution is closer to +/- 0.5 milligrams based on the cyclic loading characterization. The decrease in resolution is most likely attributable to vibrations during operation, which can be confirmed by recalibrating the system on a vibration isolation table. Calibration demonstrated that the system rigidity was 8.2372mg/um, providing a baseline of rigidity under which samples should be tested. Through the use of a cyclic loading test, a warm up time of at least 2 hours was determined to prevent drift between data point and repeated tests.

The MFDS system was successful in characterizing the force displacement curve from a 4.4mm^2 silicon diaphragm, producing approximately 200nm of displacement for 10 milligrams (98uN) of force applied. Theoretical equations confirmed that the testing results fall within the deflection limits of a 4.4mm^2 diaphragm, verifying that the results are reasonable.

With additional transducer characterization (through system isolation, higher resolution analytical balance), the system will be better able to serve the needs of those working in the Microfabrication Lab to quantify the physical properties of MEMS devices.

REFERENCES

1. Microelectromechanical Systems (National Research Council, 1997)
2. S. Meredith, M.S. thesis, California Polytechnic State University- San Luis Obispo, 2009.
3. R. E. Hummel, *Electronic Properties of Materials* (Springer Science+Business Media, New York, 2001), pp. 149-152.
4. J. W. Judy, Smart Materials and Structures **10,** 1115-1134 (2001).
5. S. A. Campbell, *Fabricantion Engineering at the Micro and Nanoscale* (Oxford University Press, New York, 2008), pp. 555-593.
6. Multitest, *Strategic Approaches for MEMS Test* (multitest.com) pp.1-18.
7. P. Waters, Ph.D. dissertation, University of South Florida, 2008.
8. "Model 5948 MicroTester for Small-Scale Low-Force Testing up to 2 kN," http://www.instron.us/wa/product/MicroTester-System-for-Low-Force-Static.aspx (10 May 2013).
9. T. Hsu, *MEMS & Microsystems Design and Manufacture* (McGraw-Hill, New York, 2002), pp. 96-162.

APPENDIX I

C:\Users\microfab\Documents\GitHub\...\MFDMInterface\MFDMInterface\StageSerialCom.cs 1

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.IO.Ports;
using System.IO;

namespace MFDMInterface
{
    class StageSerialCom
    {
        const int BAUD = 19200;

        SerialPort StagePort;

        public StageSerialCom(string comPort)
        {
            StagePort = new SerialPort(comPort);
            StagePort.BaudRate = BAUD;
            StagePort.Parity = Parity.None;
            StagePort.DataBits = 8;
            StagePort.StopBits = StopBits.One;
            StagePort.Handshake = Handshake.XOnXOff;

            StagePort.Open();

            SetUnits(1, 0);
            SetUnits(2, 0);
            SetResolution(1, 1);
            SetResolution(2, 1);

            SetMaxAccDec(1, 50000);
            SetMaxAccDec(2, 50000);

            SetAcceleration(1, 50000);
            SetAcceleration(2, 50000);
            SetDeceleration(1, 50000);
            SetDeceleration(2, 50000);

            SetMaxVelocity(1, 141374);
            SetMaxVelocity(2, 141374);

            SetVelocity(1, 70000);
            SetVelocity(2, 70000);
        }

        public void SendCommand(int stageNum, string command, int setting)
        {
            StagePort.Write(stageNum + command + setting + "\r");
        }

        public void SetMaxVelocity(int stageNum, int maxVelocity)
        {
            SendCommand(stageNum, "VU", maxVelocity);
        }

        public void SetVelocity(int stageNum, int velocity)
        {
           SendCommand(stageNum, "VA", velocity);
        }

        public void SetResolution(int stageNum, int resolution)
        {
            SendCommand(stageNum, "SU", resolution);
        }
```

```
        public void SetAcceleration(int stageNum, int acceleration)
        {
            SendCommand(stageNum, "AC", acceleration);
        }

        public void SetDeceleration(int stageNum, int deceleration)
        {
            SendCommand(stageNum, "AG", deceleration);
        }

        public void MoveRelative(int stageNum, int units)
        {
            SendCommand(stageNum, "PR", units);
        }

        public void SetMaxAccDec(int stageNum, int accDec)
        {
            SendCommand(stageNum, "AU", accDec);
        }

        public void SetUnits(int stageNum, int units)
        {
            SendCommand(stageNum, "SN", units);
        }
    }
}
```

```
using System;
using System.Threading;
using System.Collections.Generic;
using System.IO.Ports;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace MFDMInterface
{
    class TestingUtility
    {
        private static StageController MovementController;
        private static SerialPort KeithleyPort;
        public static Boolean KeepRunningTest;

        public delegate void DataUpdateDelegate(float displacement, float voltage);

        private class TestRunner
        {
            private DataUpdateDelegate GraphUpdate;
            private float StopVoltage;
            private int ReadingDelay;
            private int StepSize;
            private string OutFile;

            public TestRunner(DataUpdateDelegate graphUpdate, float stopVoltage, int readingDelay, int
    stepSize, string outFile)
            {
                GraphUpdate = graphUpdate;
                StopVoltage = stopVoltage;
                ReadingDelay = readingDelay;
                StepSize = stepSize;
                OutFile = outFile;
            }

            public void RunTest()
            {
                float curVoltage;
                float displacement = 0;
                string strVolt;

                System.IO.StreamWriter file = new System.IO.StreamWriter(@"C:\Users\microfab\Desktop\" +
    OutFile);
                file.Write("Date: " + DateTime.Now + " Stop Voltage: " + StopVoltage + " (v) Reading Delay:
     " + ReadingDelay + " (ms) Step Size: " + StepSize + " (en)\n");

                do
                {
                    MovementController.ZNegative(StepSize);
                    displacement += StepSize;
                    System.Threading.Thread.Sleep(ReadingDelay);

                    KeithleyPort.Write(":READ?\r");
                    strVolt = KeithleyPort.ReadLine();
                    file.Write(displacement + "," + strVolt);
                    curVoltage = float.Parse(strVolt, System.Globalization.CultureInfo.InvariantCulture);
                    GraphUpdate(displacement, curVoltage);
                } while (curVoltage >= StopVoltage && KeepRunningTest);
                file.Close();
                KeithleyPort.Close();
            }
        }

        public TestingUtility(StageController stageCont, string keithleyPort)
        {
```

```
            MovementController = stageCont;

            KeithleyPort = new SerialPort(keithleyPort);
            KeithleyPort.BaudRate = 19200;
            KeithleyPort.Parity = Parity.None;
            KeithleyPort.DataBits = 8;
            KeithleyPort.StopBits = StopBits.One;
            KeithleyPort.Handshake = Handshake.None;
        }

        public void SetupKeithley()
        {
            KeithleyPort.Write("*RST\r");
            KeithleyPort.Write(":SENS:FUNC 'VOLT'\r");
            KeithleyPort.Write(":SENS:RES:NPLC 1\r");
            KeithleyPort.Write(":SENS:RES:MODE MAN\r");
            KeithleyPort.Write(":SENS:VOLT:PROT 100\r");
            KeithleyPort.Write(":SOUR:CLE:AUTO ON\r");
            KeithleyPort.Write(":TRIG:COUN 1\r");
            KeithleyPort.Write(":FORM:ELEM VOLT\r");
            KeithleyPort.Write(":OUTP ON\r");
            KeithleyPort.Write(":READ?\r");
        }

        public void RunTest(DataUpdateDelegate graphUpdate, float stopVoltage, int readingDelay, int       ↙
    stepSize, string outFile)
        {
            TestRunner runner = new TestRunner(graphUpdate, stopVoltage, readingDelay, stepSize, outFile);
            Thread tThread = new Thread(new ThreadStart(runner.RunTest));

            KeepRunningTest = true;
            tThread.Start();
        }

        public void MoveToTestPosition(int xOffset, int yOffset)
        {
            MovementController.XPositive(-xOffset);
            MovementController.YPositive(-yOffset);
        }

        public void OpenPort()
        {
            KeithleyPort.Open();
            SetupKeithley();
        }

        public void ClosePort()
        {
            KeithleyPort.Close();
        }
    }
}
```

C:\Users\microfab\Documents\GitHub\MicroforceDisplacementMachine\MFDMInterface\MFDMInterface\YAxis.cs 1

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace MFDMInterface
{
    class YAxis : StageAxis
    {
        private int YChannel;

        public YAxis(StageSerialCom comm, int stageChannel)
            : base(comm)
        {
            YChannel = stageChannel;
        }

        public void Move(int uM)
        {
            SerialComm.MoveRelative(YChannel, uM);
        }
    }
}
```

C:\Users\microfab\Documents\GitHub\MicroforceDisplacementMachine\MFDMInterface\MFDMInterface\ZAxis.cs 1

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace MFDMInterface
{
    class ZAxis : StageAxis
    {
        private int ZChannel;

        public ZAxis(StageSerialCom comm, int stageChannel)
            : base(comm)
        {
            ZChannel = stageChannel;
        }

        public void Move(int uM)
        {
            SerialComm.MoveRelative(ZChannel, uM);
        }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace MFDMInterface
{
    class StageController : INotifyPropertyChanged
    {
        public static string XPort = "COM4";
        public static string YZPort = "COM5";

        public static int XLChannel = 2;
        public static int XRChannel = 1;
        public static int ZChannel = 2;
        public static int YChannel = 1;

        public int ZResolution
        {
            get { return ZRes; }
            set { ZRes = value; }
        }

        public int XResolution
        {
            get { return XRes; }
            set { XRes = value; }
        }

        public int YResolution
        {
            get { return YRes; }
            set { YRes = value; }
        }

        public int XMovement
        {
            get { return _XMoveSinceLastRst; }
            private set
            {
                _XMoveSinceLastRst = value;
                FormattedXMovement = "Actual X Offset: " + value + " (en)";
            }
        }

        public int YMovement
        {
            get { return _YMoveSinceLastRst; }
            private set
            {
                _YMoveSinceLastRst = value;
                FormattedYMovement = "Actual Y Offset: " + value + " (en)";
            }
        }

        public string FormattedXMovement
        {
            get { return _FormattedX; }
            private set
            {
                _FormattedX = value;
                OnPropertyChanged("FormattedXMovement");
            }
        }
```

```
        public string FormattedYMovement
        {
            get { return _FormattedY; }
            private set
            {
                _FormattedY = value;
                OnPropertyChanged("FormattedYMovement");
            }
        }

        StageSerialCom XCommunicator;
        StageSerialCom YZCommunicator;

        private XAxis XAxisController;
        private YAxis YAxisController;
        private ZAxis ZAxisController;

        public static int OneMillimeter = 28275;
        public static int FiveHundredMicrons = 14137;
        public static int OneMicron = 28;
        public static int HalfMicron = 14;

        private int XRes = FiveHundredMicrons;
        private int YRes = FiveHundredMicrons;
        private int ZRes = FiveHundredMicrons;

        private int _XMoveSinceLastRst;
        private int _YMoveSinceLastRst;

        private string _FormattedX;
        private string _FormattedY;

        public event PropertyChangedEventHandler PropertyChanged;

        public StageController()
        {
            XCommunicator = new StageSerialCom(XPort);
            YZCommunicator = new StageSerialCom(YZPort);

            XAxisController = new XAxis(XCommunicator, XLChannel, XRChannel);
            YAxisController = new YAxis(YZCommunicator, YChannel);
            ZAxisController = new ZAxis(YZCommunicator, ZChannel);
        }

        public void ResetMovement()
        {
            XMovement = 0;
            YMovement = 0;
        }

        public void ZNegative(int encoderUnits)
        {
            ZAxisController.Move(encoderUnits);
        }

        public void ZPositive(int encoderUnits)
        {
            ZAxisController.Move(-1 * encoderUnits);
        }

        public void ZNegativeAuto()
        {
            ZNegative(HalfMicron);
        }
```

```
        public void ZPositiveAuto()
        {
            ZPositive(HalfMicron);
        }

        public void ZNegative()
        {
            ZNegative(ZResolution);
        }

        public void ZPositive()
        {
            ZPositive(ZResolution);
        }

        public void XNegative(int encoderUnits)
        {
            XMovement += -1 * encoderUnits;
            XAxisController.Move(-1 * encoderUnits);
        }

        public void XPositive(int encoderUnits)
        {
            XMovement += encoderUnits;
            XAxisController.Move(encoderUnits);
        }

        public void XNegative()
        {
            XNegative(XResolution);
        }

        public void XPositive()
        {
            XPositive(XResolution);
        }

        public void YNegative(int encoderUnits)
        {
            YMovement += -1 * encoderUnits;
            YAxisController.Move(-1 * encoderUnits);
        }

        public void YPositive(int encoderUnits)
        {
            YMovement += encoderUnits;
            YAxisController.Move(encoderUnits);
        }

        public void YNegative()
        {
            YNegative(YResolution);
        }

        public void YPositive()
        {
            YPositive(YResolution);
        }

        protected virtual void OnPropertyChanged(string property)
        {
            if (PropertyChanged != null)
                PropertyChanged(this, new PropertyChangedEventArgs(property));
        }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Windows.Forms.DataVisualization.Charting;

namespace MFDMInterface
{
    public partial class Form1 : Form
    {
        private StageController MovementController;
        private CalibrationUtility CalUtill;
        private TestingUtility TestUtil;

        delegate void UpdateGraphCallback(float displacement, float voltage);

        public Form1()
        {
            InitializeComponent();
            MovementController = new StageController();
            CalUtill = new CalibrationUtility(MovementController, "COM6", "COM3"); //THIS IS THE LINE (Move ↙
    ...roller, Balance, Keithley)
            TestUtil = new TestingUtility(MovementController, "COM3");
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            assXOffLabel.Text = "Assumed X Offset: " + CalUtill.XOffset + " (en)";
            assYOffLabel.Text = "Assumed Y Offset: " + CalUtill.YOffset + " (en)";

            forceLabel.Text = "Force: -- (mg)";
            voltageLabel.Text = "Voltage: -- (v)";

            actXOffLabel.DataBindings.Add(new Binding("Text", MovementController, "FormattedXMovement"));
            actYOffLabel.DataBindings.Add(new Binding("Text", MovementController, "FormattedYMovement"));
            MovementController.ResetMovement();

            verticalStepLabel.Text = "Vertical Step Size: " + verticalStepBar.Value + " (en)";
            stepSizeLabel.Text = "Horizontal Step Size: " + XYStepBar.Value + " (en)";

            forceDisplacementChart.Series[0].ChartType = System.Windows.Forms.DataVisualization.Charting.   ↙
    SeriesChartType.Line;
            forceDisplacementChart.Series[0].Name = "Force vs. Displacement";
            forceDisplacementChart.ChartAreas[0].AxisX.Title = "Displacement (en)";
            forceDisplacementChart.ChartAreas[0].AxisY.Title = "Force (mg)";

            if (!icImagingControl1.DeviceValid)
            {
                icImagingControl1.ShowDeviceSettingsDialog();

                if (!icImagingControl1.DeviceValid)
                {
                    MessageBox.Show("No device was selected.", "Grabbing an Image",
                                    MessageBoxButtons.OK, MessageBoxIcon.Information);
                    this.Close();
                }
                int width = icImagingControl1.Width;
                int height = icImagingControl1.Height;
                icImagingControl1.LiveDisplayDefault = false;
                icImagingControl1.LiveDisplayHeight = icImagingControl1.Height;
                icImagingControl1.LiveDisplayWidth = icImagingControl1.Width;
```

```
            icImagingControl1.OverlayBitmap.Enable = true;
            icImagingControl1.OverlayBitmap.ColorMode = TIS.Imaging.OverlayColorModes.Color;

            icImagingControl1.LiveDisplayZoomFactor = (float)sldZoom.Value / 10.0f;
            lblZoomPercent.Text = (sldZoom.Value * 10).ToString() + "%";
            icImagingControl1.LiveDisplayPosition = new Point((-1 * icImagingControl1.LiveDisplayWidth ↙
/ 2) + icImagingControl1.Width / 2,
                (-1 * icImagingControl1.LiveDisplayHeight / 2) + icImagingControl1.Height / 2);

            icImagingControl1.LiveStart();
            DrawCrosshairs();
        }
    }

    public void UpdateGraph(float displacement, float voltage)
    {
        if (this.forceDisplacementChart.InvokeRequired) {
            UpdateGraphCallback d = new UpdateGraphCallback(UpdateGraph);
            this.Invoke(d, new object[] {displacement, voltage});
        } else {
            float force = -56.625f * voltage ; //********************************** FORCE EQUATION *** ↙
************************************\\
            forceDisplacementChart.Series[0].Points.AddXY(displacement, force);
            forceLabel.Text = "Force: " + force.ToString("n4") + " (mg)";
            voltageLabel.Text = "Voltage: " + voltage.ToString("n4") + " (v)";
        }
    }

    private void LeftButton_Click(object sender, EventArgs e)
    {
        MovementController.XNegative(XYStepBar.Value);
    }

    private void UpButton_Click(object sender, EventArgs e)
    {
        MovementController.YPositive(XYStepBar.Value);
    }

    private void RightButton_Click(object sender, EventArgs e)
    {
        MovementController.XPositive(XYStepBar.Value);
    }

    private void DownButton_Click(object sender, EventArgs e)
    {
        MovementController.YNegative(XYStepBar.Value);
    }

    private void XYStepBar_Scroll(object sender, EventArgs e)
    {
        stepSizeLabel.Text = "Step Size: " + XYStepBar.Value + " (en)";
    }

    private void sldZoom_Scroll(object sender, EventArgs e)
    {
        if (icImagingControl1.LiveDisplayDefault == false)
        {
            icImagingControl1.LiveDisplayZoomFactor = (float)sldZoom.Value / 10.0f;
            lblZoomPercent.Text = (sldZoom.Value * 10).ToString() + "%";
            int centerX = (-1 * icImagingControl1.LiveDisplayWidth / 2) + icImagingControl1.Width / 2;
            int centerY = (-1 * icImagingControl1.LiveDisplayHeight / 2) + icImagingControl1.Height / 2 ↙
;
            icImagingControl1.LiveDisplayPosition = new Point(centerX, centerY);

            DrawCrosshairs();
```

```
            }
            else
            {
                MessageBox.Show("The zoom factor can only be set" + "\n" + "if LiveDisplayDefault returns
    False!");
            }
        }

        private void DrawCrosshairs()
        {
            int width = icImagingControl1.ImageWidth;
            int height = icImagingControl1.ImageHeight;
            int crosshairLength = (icImagingControl1.ImageWidth / icImagingControl1.LiveDisplayWidth) * 50;

            icImagingControl1.OverlayBitmap.Fill(icImagingControl1.OverlayBitmap.DropOutColor);
            for (int i = Math.Max(0, 7 - sldZoom.Value); i >= 0; i--)
            {
                icImagingControl1.OverlayBitmap.DrawLine(Color.Red, width / 2 - crosshairLength, height / 2
    , width / 2 + crosshairLength, height / 2);
                icImagingControl1.OverlayBitmap.DrawLine(Color.Red, width / 2, height / 2 - crosshairLength
    , width / 2, height / 2 + crosshairLength);

                icImagingControl1.OverlayBitmap.DrawLine(Color.Red, width / 2 - crosshairLength, height / 2
     + i, width / 2 + crosshairLength, height / 2 + i);
                icImagingControl1.OverlayBitmap.DrawLine(Color.Red, width / 2 + i, height / 2 -
    crosshairLength, width / 2 + i, height / 2 + crosshairLength);

                icImagingControl1.OverlayBitmap.DrawLine(Color.Red, width / 2 - crosshairLength, height / 2
     - i, width / 2 + crosshairLength, height / 2 - i);
                icImagingControl1.OverlayBitmap.DrawLine(Color.Red, width / 2 - i, height / 2 -
    crosshairLength, width / 2 - i, height / 2 + crosshairLength);
            }
        }

        private void probeUp_Click(object sender, EventArgs e)
        {
            MovementController.ZPositive(verticalStepBar.Value);
        }

        private void probeDown_Click(object sender, EventArgs e)
        {
            MovementController.ZNegative(verticalStepBar.Value);
        }

        private void bcCal_Click(object sender, EventArgs e)
        {
            CalUtill.OpenPorts();
            forceDisplacementChart.Series[0].Points.Clear();
            CalUtill.GenerateBalanceKeithleyCalibrationData(new CalibrationUtility.DataUpdateDelegate
    (UpdateGraph), float.Parse(stopValue.Text), int.Parse(readingDelay.Text), int.Parse(stepSize.Text),
    outFile.Text);
        }

        private void verticalStepBar_Scroll(object sender, EventArgs e)
        {
            verticalStepLabel.Text = "Vertical Step Size: " + verticalStepBar.Value + " (en)";
        }

        private void calButton_Click(object sender, EventArgs e)
        {
            MovementController.ResetMovement();
            MovementController.XPositive(CalUtill.XOffset);
            MovementController.YPositive(CalUtill.YOffset);
        }

        private void setButton_Click(object sender, EventArgs e)
```

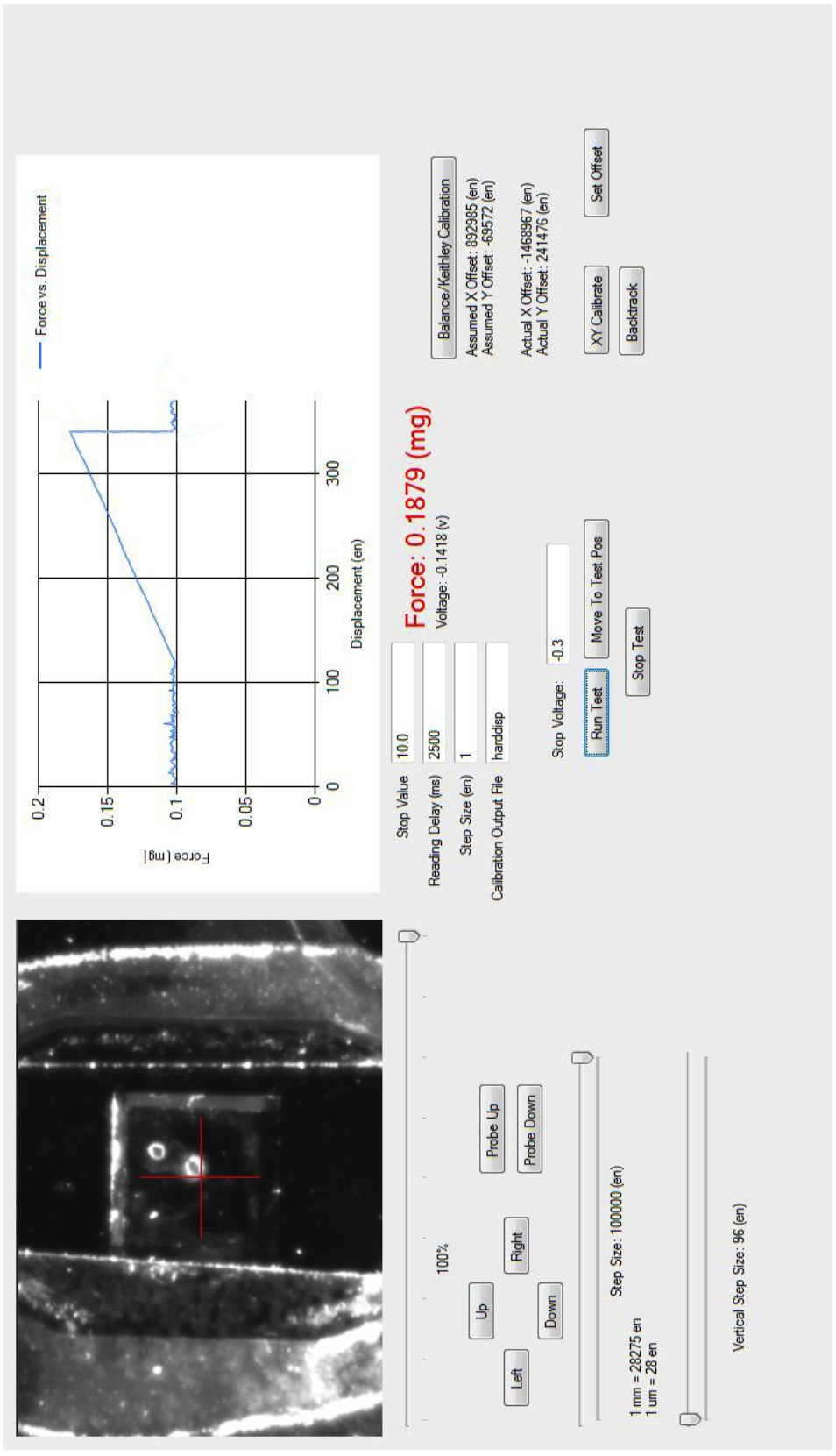
```
        {
            CalUtill.XOffset = MovementController.XMovement;
            CalUtill.YOffset = MovementController.YMovement;
            assXOffLabel.Text = "Assumed X Offset: " + CalUtill.XOffset + " (en)";
            assYOffLabel.Text = "Assumed Y Offset: " + CalUtill.YOffset + " (en)";
            actXOffLabel.Text = "Actual X Offset: " + CalUtill.XOffset + " (en)";
            actYOffLabel.Text = "Actual Y Offset: " + CalUtill.YOffset + " (en)";
        }

        private void backtrackButton_Click(object sender, EventArgs e)
        {
            MovementController.XPositive(-CalUtill.XOffset);
            MovementController.YPositive(-CalUtill.YOffset);
        }

        private void runButton_Click(object sender, EventArgs e)
        {
            TestUtil.OpenPort();
            forceDisplacementChart.Series[0].Points.Clear();
            TestUtil.RunTest(new TestingUtility.DataUpdateDelegate(UpdateGraph), float.Parse
    (stopVoltageText.Text), int.Parse(readingDelay.Text), int.Parse(stepSize.Text), outFile.Text);
        }

        private void testPosition_Click(object sender, EventArgs e)
        {
            TestUtil.MoveToTestPosition(CalUtill.XOffset, CalUtill.YOffset);
        }

        private void stopButton_Click(object sender, EventArgs e)
        {
            TestingUtility.KeepRunningTest = false;
            CalibrationUtility.KeepRunningTest = false;
            forceLabel.Text = "Force: -- (mg)";
            voltageLabel.Text = "Voltage: -- (v)";
        }

    }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.IO.Ports;
using System.IO;
using System.Threading;

namespace MFDMInterface
{
    class CalibrationUtility
    {
        private static StageController MovementController;
        private static SerialPort BalancePort;
        private static SerialPort KeithleyPort;
        public static Boolean KeepRunningTest;

        public int XOffset;
        public int YOffset;

        public delegate void DataUpdateDelegate(float displacement, float voltage);

        private class CalibrationRunner
        {
            private DataUpdateDelegate GraphUpdate;
            private float StopValue;
            private int ReadingDelay;
            private int StepSize;
            private string OutFile;

            public CalibrationRunner(DataUpdateDelegate graphUpdate, float stopVoltage, int readingDelay,
    int stepSize, string outFile)
            {
                GraphUpdate = graphUpdate;
                StopValue = stopVoltage;
                ReadingDelay = readingDelay;
                StepSize = stepSize;
                OutFile = outFile;
            }

            public void RunTest()
            {
                float curPressure;
                char[] splitChars = { ' ' };
                string result;
                string[] splitResult;
                float displacement = 0;
                string strVolt;

                System.IO.StreamWriter file = new System.IO.StreamWriter(@"C:\Users\microfab\Desktop\" +
    OutFile);

                do
                {
                    MovementController.ZNegative(StepSize);
                    displacement += StepSize;
                    System.Threading.Thread.Sleep(ReadingDelay);
                    BalancePort.Write("!KP\r");
                    result = BalancePort.ReadLine();
                    splitResult = result.Split(splitChars);

                    foreach (string str in splitResult)
                    {
                        if (str.Contains('.'))
                        {
```

```
                            result = str;
                            break;
                        }
                    }
                    curPressure = float.Parse(result, System.Globalization.CultureInfo.InvariantCulture);
                    KeithleyPort.Write(":READ?\r");
                    strVolt = KeithleyPort.ReadLine();
                    file.Write(curPressure + "," + strVolt);
                } while (curPressure <= StopValue);
                file.Close();
                KeithleyPort.Close();
                BalancePort.Close();
            }
        }

        public CalibrationUtility(StageController stageCont, string balancePort, string keithleyPort)
        {
            XOffset = 892985;
            YOffset = -69572;

            MovementController = stageCont;

            BalancePort = new SerialPort(balancePort);
            BalancePort.BaudRate = 9600;
            BalancePort.Parity = Parity.None;
            BalancePort.DataBits = 8;
            BalancePort.StopBits = StopBits.One;
            BalancePort.Handshake = Handshake.None;

            KeithleyPort = new SerialPort(keithleyPort);
            KeithleyPort.BaudRate = 19200;
            KeithleyPort.Parity = Parity.None;
            KeithleyPort.DataBits = 8;
            KeithleyPort.StopBits = StopBits.One;
            KeithleyPort.Handshake = Handshake.None;
        }

        public void SetupKeithley()
        {
            KeithleyPort.Write("*RST\r");
            KeithleyPort.Write(":SENS:FUNC 'VOLT'\r");
            KeithleyPort.Write(":SENS:RES:NPLC 1\r");
            KeithleyPort.Write(":SENS:RES:MODE MAN\r");
            KeithleyPort.Write(":SENS:VOLT:PROT 100\r");
            KeithleyPort.Write(":SOUR:CLE:AUTO ON\r");
            KeithleyPort.Write(":TRIG:COUN 1\r");
            KeithleyPort.Write(":FORM:ELEM VOLT\r");
            KeithleyPort.Write(":OUTP ON\r");
            KeithleyPort.Write(":READ?\r");
        }

        public void OpenPorts()
        {
            KeithleyPort.Open();
            SetupKeithley();
            BalancePort.Open();
        }

        public void ClosePorts()
        {
            KeithleyPort.Close();
            BalancePort.Close();
        }

        public void GenerateBalanceKeithleyCalibrationData(DataUpdateDelegate graphUpdate, float stopValue, ↙
    int readingDelay, int stepSize, string outFile)
```

```
        {
            CalibrationRunner runner = new CalibrationRunner(graphUpdate, stopValue, readingDelay, stepSize ↙
    , outFile);
            Thread tThread = new Thread(new ThreadStart(runner.RunTest));

            KeepRunningTest = true;
            tThread.Start();
        }
    }
}
```

APPENDIX II


Force vs. Displacement
Force (mg)
Displacement (en)
0.2
0.15
0.1
0.05
0
0
100
200
300
Force: 0.1879 (mg)
Voltage: -0.1418 (v)
Balance/Keithley Calibration
Assumed X Offset: 892985 (en)
Assumed Y Offset: 63572 (en)
Actual X Offset: -1468967 (en)
Actual Y Offset: 241476 (en)
Stop Value 10.0
Reading Delay (ms) 2500
Step Size (en) 1
Calibration Output File harddisp
Stop Voltage: -0.3
Run Test
Move To Test Pos
Stop Test
XY Calibrate
Backtrack
Set Offset
100%
Up
Left
Right
Down
Probe Up
Probe Down
Step Size: 100000 (en)
1 mm = 28275 en
1 um = 28 en
Vertical Step Size: 96 (en)